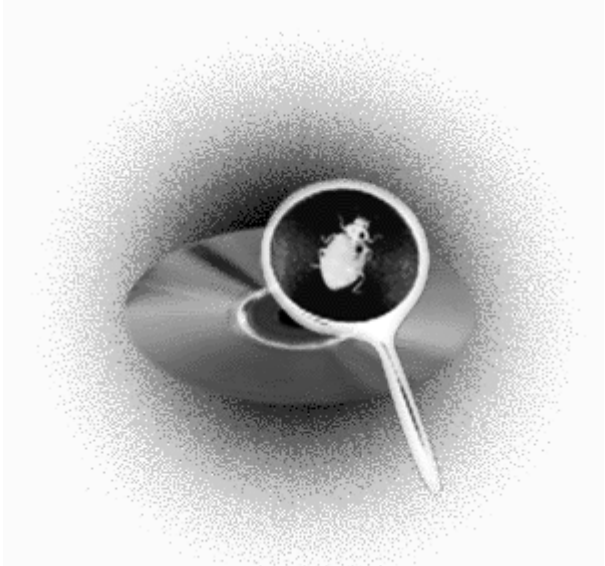
# 第三世代に突入したソフトウェアテスト



ソフトウェアテストシンポジウム四国 2010

2010/7/2（金）

電気通信大学 大学院 情報理工学研究科

総合情報学専攻 経営情報学コース

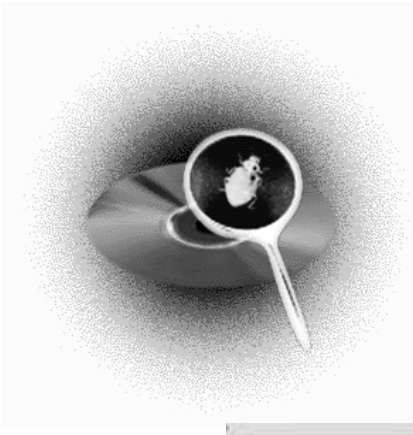
西　康晴

# 自己紹介

- 身分
  - ソフトウェア工学の研究者
    - » 電気通信大学 大学院 情報理工学研究科 総合情報学専攻 経営情報学コース
    - » ちょっと「生臭い」研究／ソフトウェアテストやプロセス改善など
  - 先日までソフトウェアのよろず品質コンサルタント
- 専門分野
  - ソフトウェアテスト／プロジェクトマネジメント／QA／ソフトウェア品質／TQM全般／教育
- 共訳書
  - 実践ソフトウェア・エンジニアリング／日科技連出版
  - 基本から学ぶソフトウェアテスト／日経BP
  - ソフトウェアテスト293の鉄則／日経BP
- もろもろ
  - TEF: テスト技術者交流会 / ASTER: テスト技術振興協会
  - WACATE: 若手テストエンジニア向けワークショップ
  - SESSAME: 組込みソフトウェア管理者技術者育成研究会
  - SQiP: 日科技連ソフトウェア品質委員会
  - 情報処理学会 ソフトウェア工学研究会 / SE教育委員会
  - ISO/IEC JTC1/SC7/WG26（ISO/IEC29119 ソフトウェアテスト）
  - 経済産業省 組込みソフトウェア開発力強化推進委員会

# TEF: Software Testing Engineer's Forum

- ソフトウェアテスト技術者交流会
  - 1998年9月に活動開始
    - » 現在1700名強の登録
    - » MLベースの議論と、たまの会合
  - http://www.swtest.jp/forum.html
  - お金は無いけど熱意はあるテスト技術者を
    無償で応援する集まり
  - 「基本から学ぶソフトウェアテスト」や
    「ソフトウェアテスト293の鉄則」の翻訳も手がける
    - » ほぼMLとWebをインフラとした珍しいオンライン翻訳チーム
  - 技術別部会や地域別勉強会が実施されている
    - » プリンタ、Web、AV、原因結果グラフなど
    - » 東京、関西、九州、東海、札幌など
    - » TestLinkというオープンソースツールの日本語化部会もある




# ASTER: Association of Software Test EngineeRing

- ソフトウェアテスト技術振興協会
  - テストを軸にして、ソフトウェア品質向上に関する教育や調査研究、普及振興を行うNPO法人
    - » 2006年4月に設立／理事・会員は無給
  - ソフトウェアテストシンポジウム（JaSST）を開催している
    - » 実行委員は手弁当／参加費は実費＋α
    - » 毎年4Qに東京で開催／今年はのべ約1500名の参加者
    - » 今年は大阪・四国・札幌・九州・東海でも開催／会場はほぼ満席
  - ソフトウェアテストの資格試験（JSTQB）を運営している
    - » Foundation Levelは8903名の受験者・4891名の合格者
    - » Advanced Level（テストマネージャ）を2010年8月に開始
  - 各地でソフトウェアテストの教育を行っている
    - » テストのスキル標準（test.SSF）をIVIAと共同で開発している
    - » 札幌や大分などでテストの教育の実績がある
    - » 勉強会の開催などを支援することで、地場の産業振興の定着を図る
  - アジア各国とテスト技術の交流（ASTA）を行っている
  - テスト開発方法論などの先端技術を研究開発している：智美塾

# SESSAME: 組込みソフトの育成研究会

- 組込みソフトウェア技術者管理者育成研究会
  - Society for Embedded Software Skill Acquisition for Managers and Engineers
  - 2000年12月に活動開始
    - » 200名強の会員／MLベースの議論と、月イチの会合
  - http://www.sessame.jp/


- 中級の技術者を10万人育てる
  - PCソフトウェアのような「そこそこ品質」ではダメ
    - » 創造性型産業において米国に劣り、コスト競争型産業でアジアに負ける
    - » ハードウェアとの協調という点で日本に勝機があるはず
  - 育成に必要なすべてを開発する
  - オープンプロダクト／ベストエフォート
    - » 文献ポインタ集、知識体系（用語集）、初級者向けテキスト、スキル標準など
    - » 7つのワークグループ：組込みMOT・演習・MISRA-C・ETSS・子供・高信頼性
  - セミナーだけでなく、講師用セミナーも実施

# SQiP：Software Quality Profession

- 名称：
  - 日本科学技術連盟・ソフトウェア品質委員会
- 目的
  - SQiPは、ソフトウェア品質技術・施策の調査・研究・教育を通じて、実践的・実証的なソフトウェア品質方法論を確立・普及することにより、ソフトウェア品質の継続的な向上を目指す
- 3つの視点
  - ソフトウェア品質・実践・普及啓蒙
- 主軸とする活動
  1. 実践的・実証的なソフトウェア品質方法論の確立
  2. ソフトウェア品質方法論の普及促進・資格認定
  3. ソフトウェア品質向上のための国際協力の推進
- 活動方針
  1. ソフトウェア品質追究の重要性訴求
  2. 日本での実践的・実証的ソフトウェア品質方法論の形式知化
  3. グローバルな視野での活動
  4. 新しい課題へのチャレンジ

SQiP
Software Quality Profession

# Safeware翻訳プロジェクト

- 2009年11月に翻訳版を刊行した
  - Nancy Leveson（MIT）著
  - 翻訳チーム：松原友夫・片平真史（JAXA）・吉岡律夫（日本機能安全）・青木美津江（電気通信大学）・西康晴（電気通信大学）
  - 翔泳社 発行


第1部　リスクの性質
- 第1章　近代社会におけるリスク
- 第2章　コンピュータとリスク
- 第3章　事故の階層的考察
- 第4章　事故の根本原因
- 第5章　ヒューマンエラーとリスク
- 第6章　自動化システムにおける人間の役割

第2部　システム安全への序論
- 第7章　システム安全の基礎
- 第8章　システム安全の基本

第3部　定義とモデル
- 第9章　用語
- 第10章　事故とヒューマンエラーモデル

第4部　セーフウェアプログラムの要素
- 第12章　システムとソフトウェア安全プロセス
- 第13章　ハザード分析
- 第14章　ハザード分析モデルと技法
- 第15章　ソフトウェアハザードと要求分析
- 第16章　安全性のための設計
- 第17章　ヒューマンマシンインタフェースの設計
- 第18章　安全性の検証

付録
- 付録A. 医療機器：Therac-25の歴史
- 付録B. 航空宇宙：アポロ13号、DC-10型機、チャレンジャー号
- 付録C　化学産業：セベソ、フリックスボロー、ボパール
- 付録D　原子炉事故：ウィンズケール、スリーマイル島、及びチェルノブイリ

# 講演の流れ

- ソフトウェアテストの世代と軸
  - ソフトウェアテストの世代
    - » 第一世代：決まっている範囲から選ぶ技術や考え方
    - » 第二世代：範囲を広げたり自ら決める技術や考え方
    - » 第三世代：全体像や本質を捉える技術や考え方
  - ソフトウェアテストの軸
    - » ① 開発ライフサイクル軸、② テストライフサイクル軸、③ テスト技術軸、④ テスト対象軸、⑤ 派生開発軸、⑥ テストスコープ軸、⑦ 品質軸
  - HAYST法はどの世代か？
- Wモデル：第三世代技術としてテストを活用するフレームワーク
  - 自工程完結と他工程配慮
  - Wモデルを支えるテスト技術
  - Wモデルのレベル
  - Wモデルの目指すべき姿

Wモデルに関する議論は、
2010年6月に開催された
ソフトウェアシンポジウム
2010のWG2での議論から
大きな薫陶を得ています。
WG2のメンバに感謝します

# ソフトウェアの信頼性は低いと言わざるを得ない

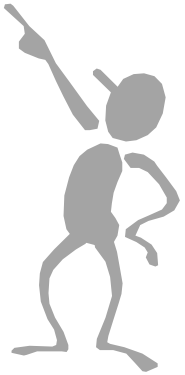
- エンタープライズ系システム
  - メガバンクの情報システム統合に伴う品質事故は記憶に新しい
  - 東京証券取引所の情報システムは度重なる品質事故
    - » 売買システムの処理増強に伴う品質事故により全銘柄で午前中の取引停止
    - » ジェイコム株誤発注事件では品質事故により400億円超の損害賠償請求訴訟
  - 羽田空港の航空管制システムの品質事故は30万人以上が足止め
  - 成功するプロジェクトはわずか31.1%（日経コンピュータ）
- 組込み系システム
  - 国内の携帯電話：65万台が回収・無償交換、131億円にのぼる損害
  - ドイツの高級車のブレーキシステム：68万台がリコール
  - 鉄道：ATCの誤作動により1ヶ月に50回以上の速度超過、19件の緊急停止措置
  - ダム：ゲートが開き、総量約2万立米の貯水が流出
- 安全性が重視されるシステムが増えていくが．．．
  - 自動車はX-by-wireになり、ITSにより道路状況を運転に反映させるようになる
  - 軌道式公共交通機関は無人になっていくかもしれない
  - 人間を殺傷する動力を持つ家庭用ロボットが、事前訓練を受けずに動かされる

# ソフトウェアテストとはどんな技術？

- いい加減な開発を助長する技術？
  - 「テストで品質を上げる」という概念を誤解している
    - » 開発上流でバグを作り込むのは当たり前、
      テストでバグを叩き出すことで品質をやっとこさっとこ確保している
    - » 品質を上げるにはテストを増やす必要があるので、
      コストや納期が余計にかかると思っている
- 開発力の向上を目指さない技術？
  - バグを叩き出すことだけが任務だと誤解している
    - » 開発の「弱点」を把握せずとにかく網羅しようとするために、
      テスト工数が膨大になり、うんざりして努力しなくなってしまう
    - » バグを報告して直してもらうだけが任務だと思っており
      開発上流の改善に寄与しないので、
      開発のたびに同じようなバグを見つける羽目になる

# ソフトウェアテストとはこんな技術

- 製品出荷時の品質や信頼性を向上させる技術
  - 少ない手間で早くたくさん危険なバグを検出できる
  - ユーザ先で発生する不具合を減少できる
- 納期遅れや残業・休出を減少させる技術
  - 製品出荷後のリスクを見積もり、闇雲に膨大なテストを防ぐことができる
- ソフトウェア全体の開発力を向上させる技術
  - 何をテストすべきかをモデリングすることで、
    俯瞰的・包括的で再利用しやすいテストを開発する
  - 質の高いテストやレビューを開発とコンカレントに進めることで、
    上流から品質のつくり込みを可能にできる
    - » バグを作り込みやすい開発の「弱点」を露わにし、
      「弱点」を重点的に狙うことで現場に負荷をかけず改善できる
    - » 手戻りを減少させるので、品質を上げるとコストが低く納期が短くなる
  - 質の高いテストをしようとすると、よい分析やよい設計、
    よい実装が必要だということに気付き努力するようになる
    - » 開発プロセスの改善を的確に進めることができる

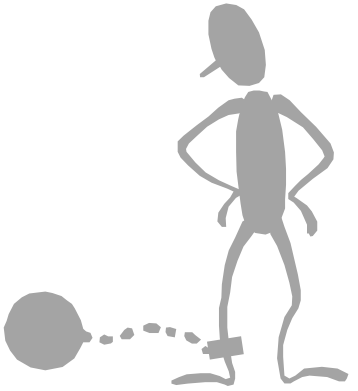


# モダン・ソフトウェア・テスティング

テストの質を高めることで上流の質が高まり、
上流の質が高まることでテストの質も高まる

# クラシック(古典的だが基本的)なテストのパラダイム

- Vモデル
  - 単体テスト／結合テスト／機能テスト／システムテスト／受け入れテスト
  - 回帰テスト
- ホワイトボックステスト／ブラックボックステスト
  - ホワイトボックステスト＝ソースコードテスト≒制御パステスト(パスカバレッジ)
  - ブラックボックステスト＝外部仕様テスト
    ≒機能網羅テスト＋境界値テスト＋デシジョンテーブルテスト
- システムテストカテゴリ
  - ボリュームテスト／ストレージテスト／高頻度テスト／ロングランテスト
  - 構成テスト／両立性テスト／データ互換性テスト
  - 操作性テスト／セキュリティテスト・・・
- プロセス無しのテスト
  - 線表引きとテスト項目作成とテスト実施と不具合報告
  - 信頼度成長曲線



# ソフトウェアテストの現状

- 適切なテストプロセスやテスト技術を導入しないと失敗する
  - テストに技術はいらない、と思うと失敗する
    - » 適当に行き当たりばったりでテストすると、漏れやムラが大量に発生する
    - » 多くのテスト技術があることを認識し、自組織に合ったテスト技術を導入する
    - » テストの設計とレビューの設計は同義である

  - 最後にまとめてテストをすればいい、と思うと失敗する
    - » テストも開発と同じように分析・設計・実装が必要であり、管理も必要である
    - » フェーズ分けをせずまとめてテストをすると、テスト漏れが発生したり、原因分析に時間がかかる
- ソフトウェアテストは開発全体のボトルネックになっている
  - 全工程の3割～9割をテストに費やしている
  - 人海戦術で納期まで徹夜を繰り返すだけ、というのが現状だろう
  - テスト技術が低いため信頼性の保証にはほど遠い、という組織が多い
    - » 技術陣だけの問題ではなく、管理層や経営層の問題でもある

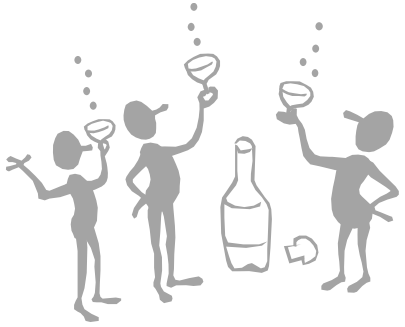
# ソフトウェアテストの世代

- 第一世代:
  - 決まっている範囲から選ぶ技術や考え方
    - » 下流でのテスト、テスト実施、制御パステスト、
      仕様やソースコード・境界値、テストケースの再利用、
      テスト対象はソフトウェア、機能の確認

  ※第一世代技術であっても、技術力を高めて
  きっちり使いこなせば品質は確保できる

ASTER / 智美塾
（2号生）で
検討中

- 第二世代:
  - 範囲を広げたり自ら決める技術や考え方
    - » 上流からのレビュー、テスト設計、リスクベースドテスト、
      モデルベースドテスト、テスト設計の再利用、
      テスト対象はシステム、品質の測定
- 第三世代:
  - 全体像や本質を捉える技術や考え方
    - » 開発全体の改善、テスト開発、テストアーキテクチャ設計、
      不具合モード、再利用しやすいテスト設計、
      テスト対象はビジネスやライフスタイル、魅力や価値の評価



# ソフトウェアテストの進化を整理するための7つの軸

① 開発ライフサイクル軸
- ／→V&V→W

② テストライフサイクル軸
- テストする→テストを設計する→テストを開発する

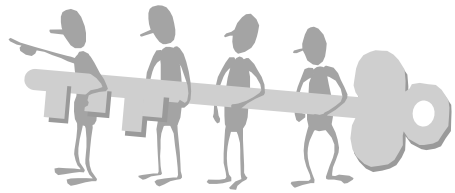
③ テスト技術軸
- 詳細設計モデルによる列挙→優先順位（リスク）→モデリングによるテスト開発

④ テスト対象軸
- 仕様とソース・境界値→設計や要求のモデル→間違えやすいところ

⑤ 派生開発軸
- ケースの再利用→テスト設計の再利用→再利用しやすいテスト設計

⑥ テストスコープ軸
- 開発対象→動作環境や人間を含めたシステム→ビジネスやライフスタイル

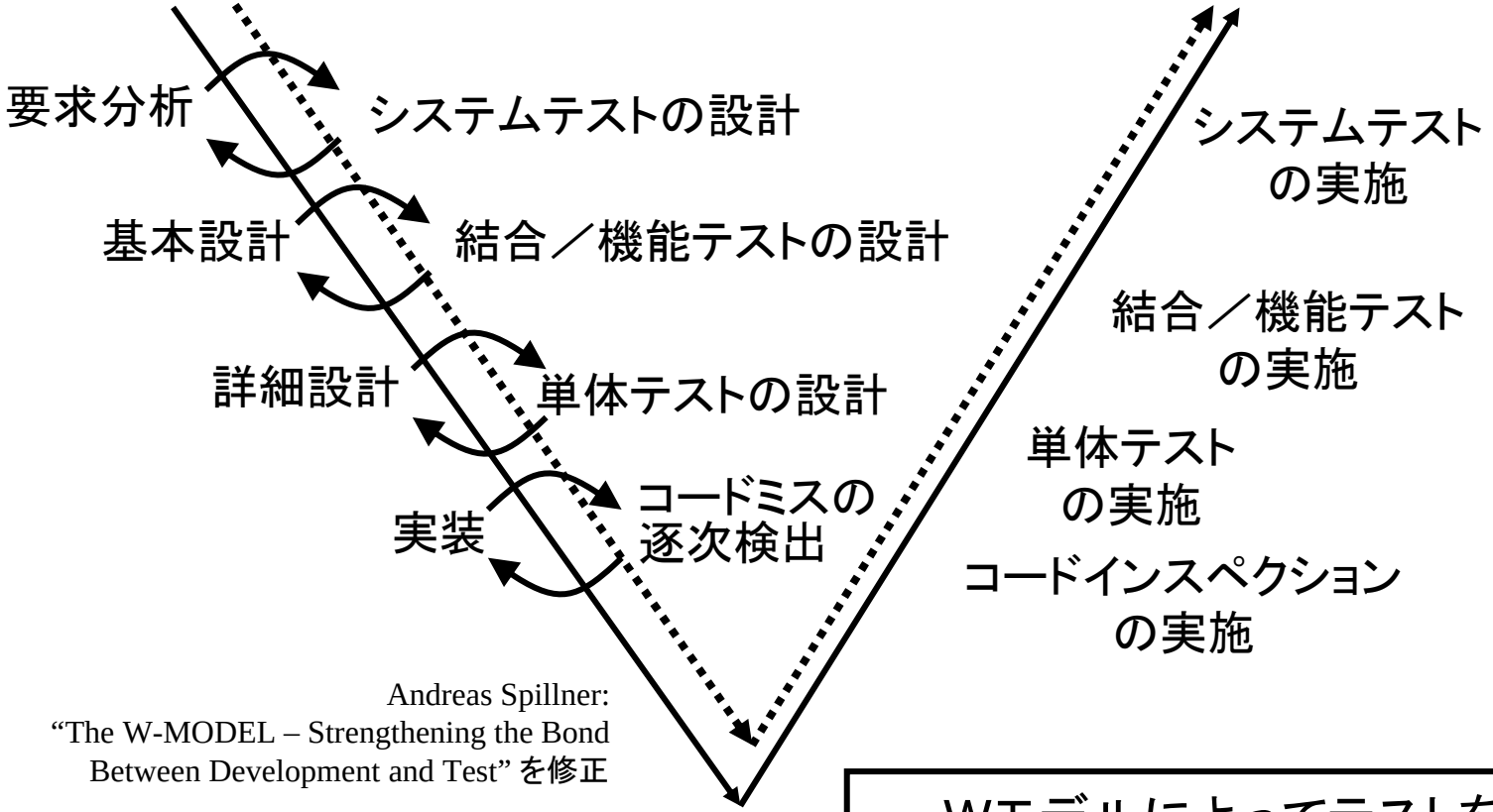
⑦ 品質軸
- 機能→品質特性→魅力



# ① 開発ライフサイクル軸

- 第一世代：／（下流でのテスト）
  - 下流になってはじめてテストを設計し実施するという考え方
    - » 下流でたくさんバグが見つかる
    - » しかし手戻りが多く発生してしまう
- 第二世代：V&V
  - テストから上流でのレビューやインスペクションに広げる考え方
    - » 上流からたくさんバグがみつかる
    - » 手戻りは少なくなるが、開発が改善されないためテストケース数は減らない
- 第三世代：Wモデル
  - テストを上流で開発し、
    最初からバグを作り込まないようにするという考え方
    - » 開発が改善されるため、
      そもそもバグが作り込まれないうえにテストケース数が減る

# 第三世代の開発ライフサイクル：Wモデル

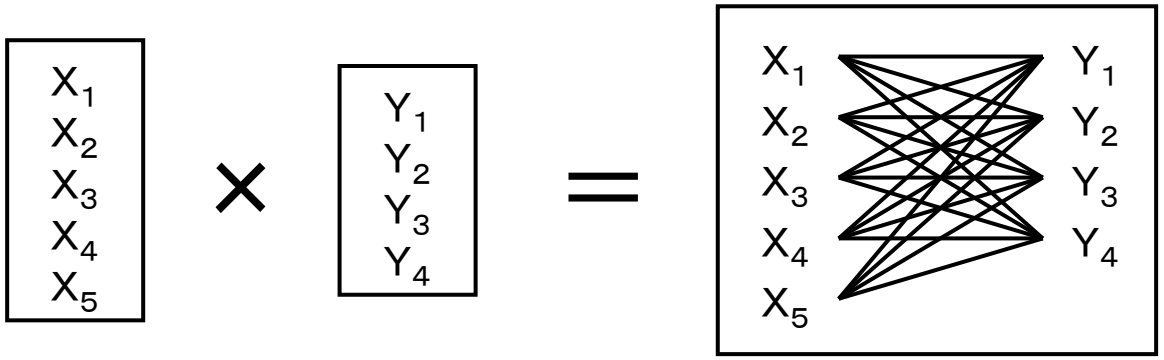



Andreas Spillner:
“The W-MODEL – Strengthening the Bond Between Development and Test” を修正

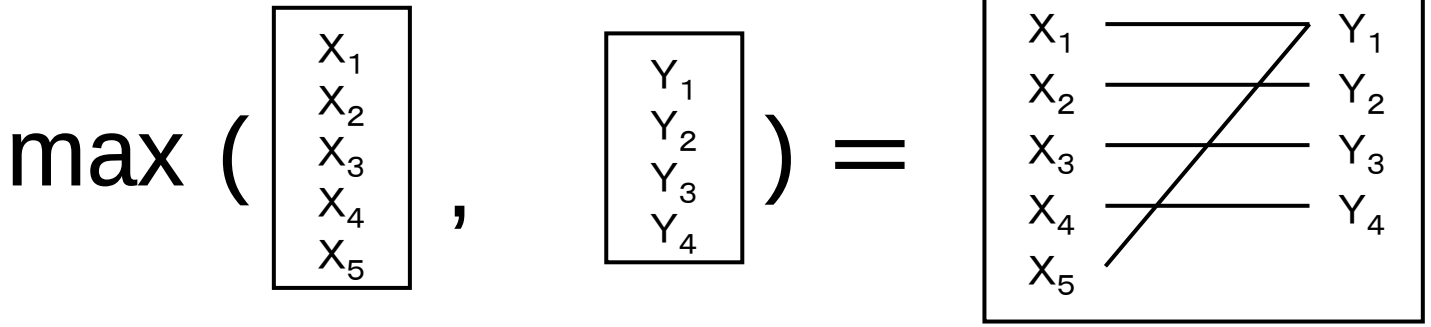
Wモデルによってテストを前倒しし、
要求や設計をスッキリさせて
そもそもバグを作り込まないようにする

# 上流での組み合わせテスト設計によるソフト設計改善

- 組み合わせ網羅テストと単一網羅テスト
  - 単機能テスト項目5件の機能Xと、単機能テスト項目4件の機能Yがある
    - » 組み合わせテスト項目数は 5×4 ＝ 20 件

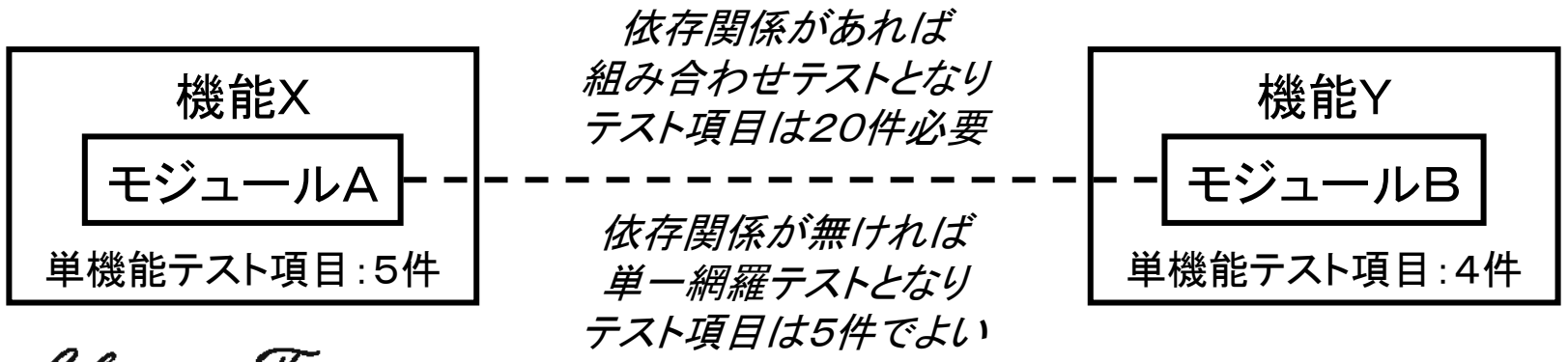



- » 単一網羅テスト項目数は max(5,4) ＝ 5 件



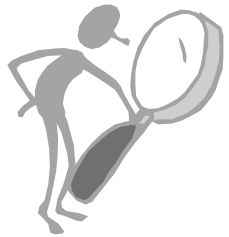
# 上流での組み合わせテスト設計によるソフト設計改善

- グレーボックスで依存関係を分析しながらテストを設計すると、機能Xと機能Yの組み合わせテストを構造的にレビューし判断して間引くことができる
  - 組み合わせが必要なのは、機能Xの処理に応じて、機能Yの処理が変わるから
    - » 機能を構成するモジュール間に依存関係があるから
  - 構造的にレビューして、依存関係が無かったり、影響伝播が抑えられていることが保証できれば、単一網羅テストで十分となる
    - » ブラックボックステストに少しホワイトボックス的観点が入るのでグレーボックステスト
    - » 実務的には完全に間引くわけではなく、優先度を落とすことになる



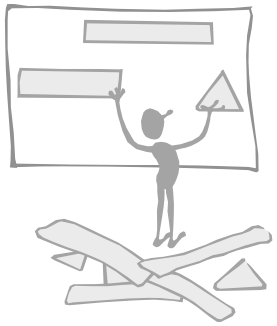
# 上流での組み合わせテスト設計によるソフト設計改善

- ソフトウェア設計とテスト設計を協調・並行して行うことで
  組み合わせが激増する「ホットスポット」を特定し
  設計そのものを改善することでテスト工数を抑える
  - 不必要な依存関係を抑えることで、
    テスト工数の爆発やTestabilityの低下につながる設計を防止する
    - » ホットスポットは設計の複雑さが集中する部分であり、
      組み合わせによるテスト工数が激増する部分である
    - » 製品設計時にモジュール性を考慮した方がよいことは皆知っているが、
      モジュール性を減らした時にどうなるかを定量的に把握したことはない
  - ホットスポットを改善しないと、派生開発の度に組み合わせテストを行う羽目になる



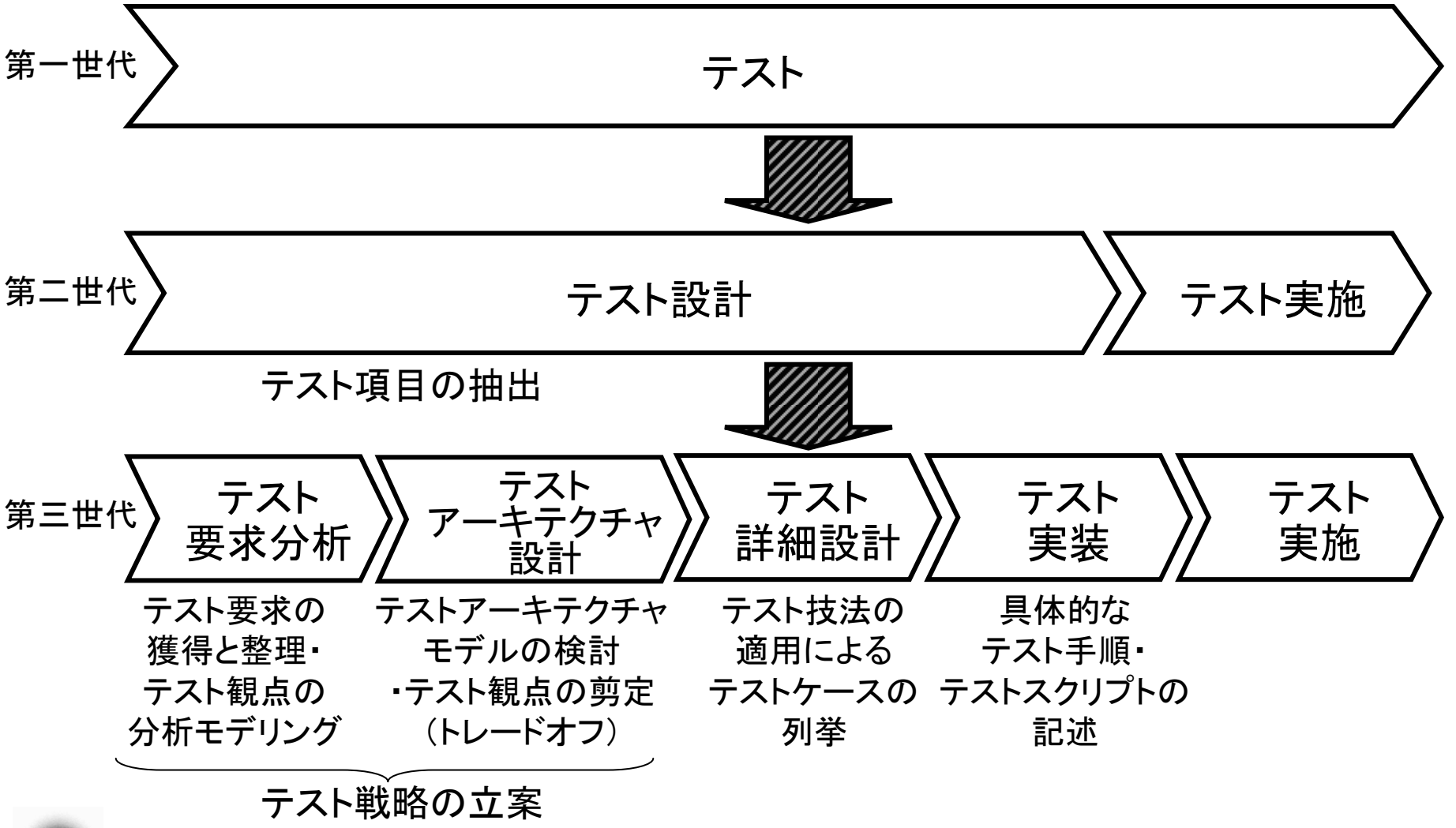
- 製品設計時にテスト工数を考慮してトレードオフを行う
  - Wモデルによって上流の設計時に
    テスト工程も考慮しながらトータルでトレードオフを行う
    - » 通常の製品設計時のトレードオフでは、テストのことなど頭にない
    - » 製品設計時には「楽だけどちょっと複雑」、テスト時には「工数爆発」
    - » 製品設計の工数がほんの少し増えても、テストで激減すればトータルで得

# ② テストライフサイクル軸



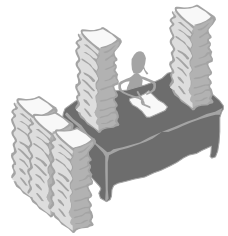
- 第一世代:テストする
  - テストを実施することだけに主眼を置く考え方
    - » 闇雲に網羅しようとする
    - » テスト計画書にテストケースが書いてある
- 第二世代:テストを設計する
  - テストケースを記述する(詳細設計する)ことに主眼を置く考え方
    - » テストの終了基準やリスクを考えようとする
    - » テスト計画書にテストアーキテクチャ(テストのレベルやタイプ)が書いてある
- 第三世代:テストを開発する
  - テストのための要求分析やテストのアーキテクチャ設計など、ソフトウェアの開発ライフサイクルと同様に高度な作業に主眼を置く考え方
    - » テストスイートを、ソフトウェアと同様の知的成果物として捉え、要求分析や設計の能力を高めようとする
    - » テスト計画書などの管理文書とテスト設計書などの技術文書が分離している

# テストライフサイクルの世代



# ③ テスト技術軸

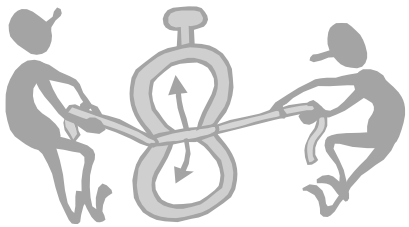
- 第一世代：詳細設計モデルによるテストケースの列挙
  - あらかじめ与えられたモデルに沿ってテストケースを列挙する技術
    - » 機能網羅テスト、制御パステスト、状態遷移テストなど
    - » とにかく網羅率を高める方向に頑張ることになる



- 第二世代：優先順位（リスク）の決定
  - 複数のモデルを組み合わせることで優先順位やリスクを決定する技術
    - » リスクベースドテスト、直交表によるテスト、all-pair法によるテストなど
    - » テストしないと何が起こるか、本当に割り切って良いのかを考えることになる
- 第三世代：モデリングによるテスト開発
  - 何をテストすべきか、からモデリングしていく技術
    - » テスト要求モデル、テストアーキテクチャ設計、テストデザインパターン
    - » どこまでをテストしてどこからをテストしないのかをきちんと整理できる

# リスクベースドテスト：優先度を決める技術

- 「もしこのテストケースを実施しなかったら
  どのくらい困ってしまうのか？」と推測し、
  困ってしまう順にテストケースを実施する方法
  - テストケースの優先度設定・間引きの方法である
    - » テストケースの進捗管理の方法でもある
    - » リスクに応じてテスト以外の手段で検出・保証することもある



- 「困ってしまう程度」を品質リスクと呼ぶ
  - 出荷後の品質低下のリスクを意味する
    - » いわゆるPM的なリスクとは意味が異なる
  - 品質リスクの高い順にテストケースを実施する
- 以下の4要素から、テストケースごとに品質リスク指数を定める
  - そのテストケースの表す状況がどのくらいの確率・頻度で発生するか
  - そのテストケースの叩く箇所にどんな欠陥がどのくらいの確率で作り込まれるか
  - その欠陥は、どのくらいの確率で不具合として露呈するか
  - その不具合の致命度はどのくらいか
- 総合的な開発力が問われる
  - 自分たちの開発力（開発把握度）が推測精度に直結する
  - きちんと中身が分かっていないと、
    リスクベースドテストは「説明無責任」の道具になる

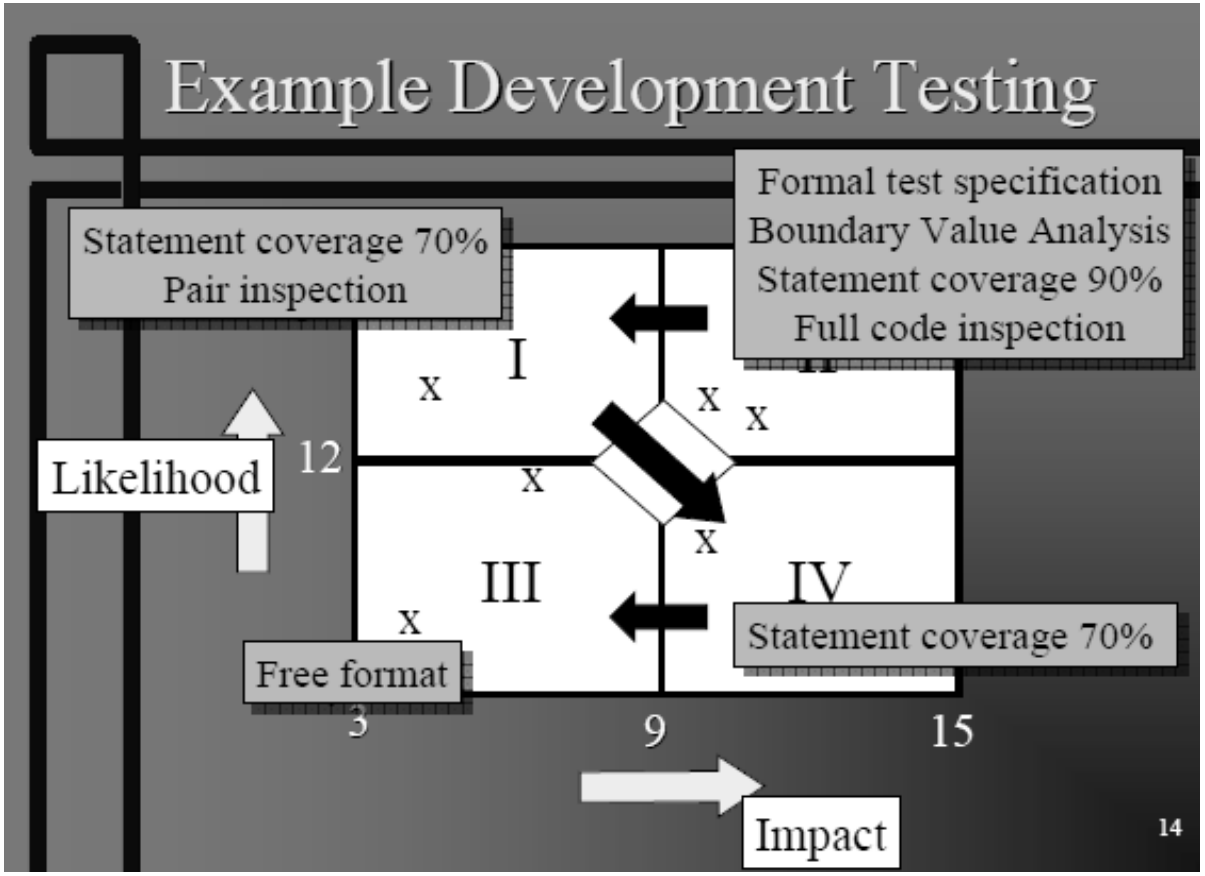
# リスクベースドテストの例

| | Likelihood | | | | | | Impact | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Complexity | New development | Interfacing | Technology | Experience level | | User imprortance | Usage intensity | Safety | |
| Item 1 | 5 | 3 | 2 | 1 | 5 | 16 | 5 | 4 | 1 | 10 |
| Item 2 | 2 | 1 | 2 | 1 | 2 | 8 | 3 | 3 | 1 | 7 |
| Item n | | | | | | | | | | |

リスクを明らかにした上で
後回しにできる品質リスクを
顧客と相談し合意する

Erik van Veenendaal:
“Risk Based Testing in Practice”
http://www.bits-chips.nl
/portal/documents/Risk%20based%20Testing
%20Erik%20van%20Veenendaal.pdf

# リスクベースドテストの例




リスクに応じて
テストの方法や
濃さを変える
こともある

Erik van Veenendaal:
"Risk Based Testing in Practice"
http://www.bits-chips.nl
/portal/documents/Risk%20based%20Testing
%20Erik%20van%20Veenendaal.pdf

# テスト要求モデルの例：HAYST法 – FV表

- FV表：Function Verification Table
  - 「ソフトウェアテストHAYST法入門」より
    - » 吉澤 正孝/秋山 浩一/仙石 太郎, 日科技連出版社, ISBN4-8171-9228-3

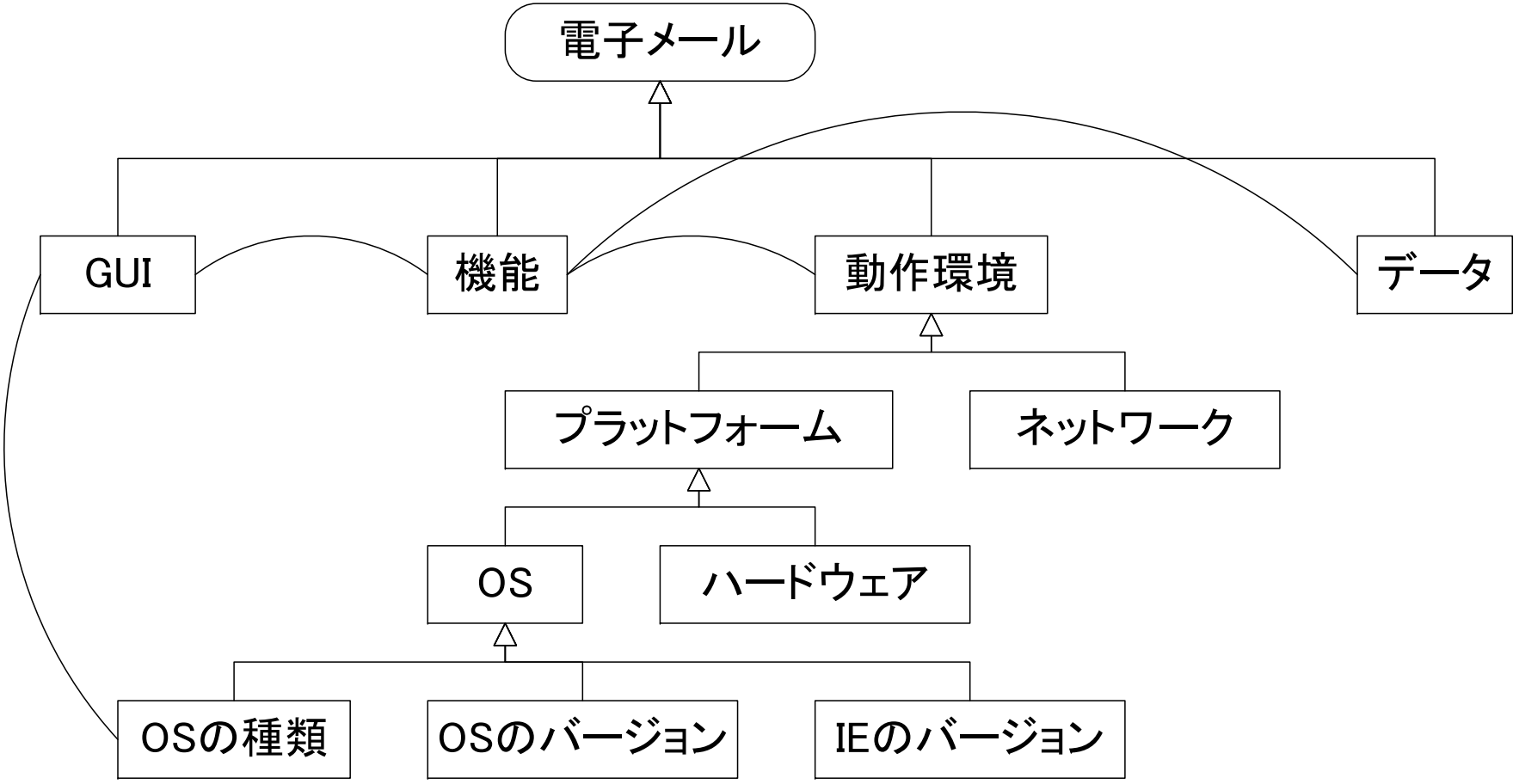
| 番号 | 機能 | 検証内容 | 使用するテスト技法 |
|---|---|---|---|
| 1 | 商品を検索する | ・希望する商品が速やかに<br>検索できること | |
| 1.1 | フリーワードで検索 | ・存在する商品名で検索できること<br>・存在しない商品名を入力したときに<br>エラーとなること | ・商品名をHAYST法で設計 |
| 1.2 | 商品カテゴリから検索 | ・リンク切れがないこと | ・総探索を自動化処理で実施 |
| 2 | 数量を選択し購入する | ・数量を入力し計算ボタンを押すことで<br>価格表示が変わること<br>・ラッピングの指示が反映されること<br>・購入ボタンで画面が遷移すること<br>・途中でブラウザが閉じられても<br>購入が継続できることを確認すること | ・HAYST法 |

# テスト要求モデルの例:ゆもつよメソッド

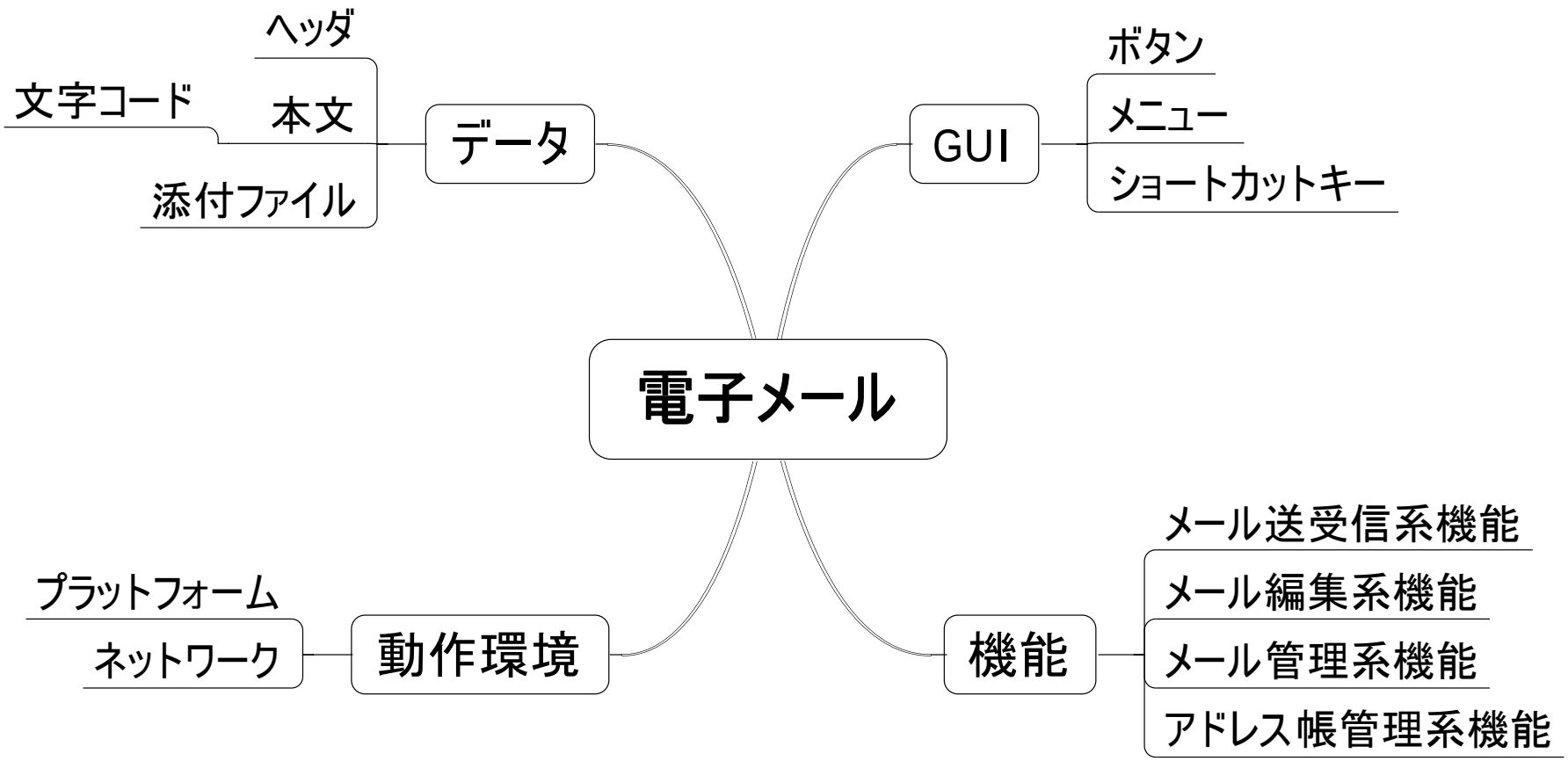
- 豆蔵の湯本さんが使っている手法

| テストカテゴリ / 機能(群) | | 入力・表示 | | 計算処理 | | | エラー処理 | 排他処理 | インタフェース | | 状態 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | アドレスの入力 | アドレスの変換 | 暗号化 | エンコード | 文字数カウント | タイムアウト | 既受信メールの編集 | SMTP | IMAP4 | サーバ未接続 | サーバ接続中 |
| メール送受信系機能群 | メール送信 | | | | | | | | | | | |
| | メール返信 | | | | | | | | | | | |
| | メール転送 | | | | | | | | | | | |
| メール編集系機能群 | 文字の入力 | | | | | | | | | | | |
| | 文字の削除 | | | | | | | | | | | |
| メール管理系機能群 | メールの削除 | | | | | | | | | | | |
| | フォルダの作成 | | | | | | | | | | | |
| | フォルダの移動 | | | | | | | | | | | |

# テスト要求モデルの例：NGTのテスト観点図

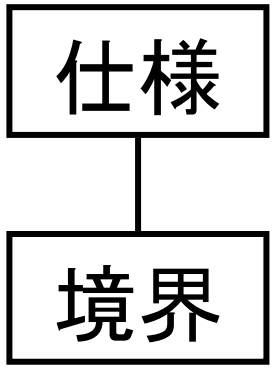


# テスト要求モデルの例：マインドマップ

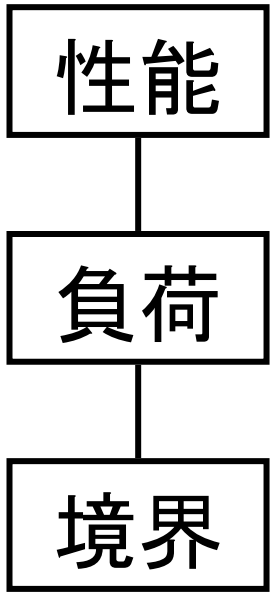


# テスト設計の上手な組織は様々なテスト観点で検討する

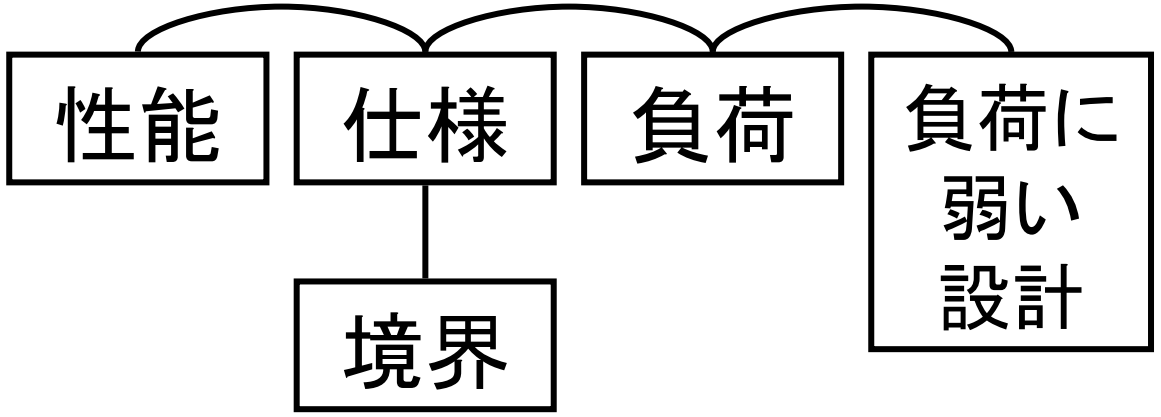
- テスト観点の列挙や詳細化を見ると、
  テスト分析・設計の成熟度が透けて見える




様々な
負荷のかけ方が
漏れる




負荷に弱い
設計部分を
的確に狙えない




技術力が必要

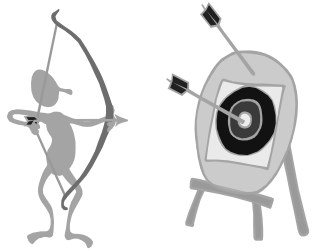
# ④ テスト対象軸

- 第一世代：仕様書とソースコード・境界値
  - 仕様書やソースコードにしたがってテストを設計する技術
    - » ブラックボックステスト・ホワイトボックステスト
    - » あらかじめ定められたテストベースを対象にすればよい

- 第二世代：設計や要求のモデル
  - ソフトウェア構造のモデルを積極的に活用してテストを設計する技術
    - » グレーボックステスト
    - » 必要ならば上流で作成されたテストベースにテストなりの修正を加えたり、上流で作成されないテストベースをリバースする必要がある
- 第三世代：間違えやすいところのモデル
  - 間違えやすいところをパターン化・モデリングしてテストを設計する技術
    - » 不具合モードなどによるピンポイントテスト、（真の意味の）探索型テスト
    - » 認知科学的、心理学的な考察が必要となる

# 不具合モードによるピンポイントテストの設計

- 不具合が作り込まれそうな「弱点」(不具合モード)を狙って
  ピンポイントでテストを設計する方法
  - 過去の不具合を蓄積・分析し、不具合が作り込まれそうな箇所を推測することで、
    ピンポイントにテスト設計を行う
    - » よく発生する、同じようなメカニズムで作り込まれたと思われる
      不具合を集めて分析することで、そのメカニズムを推定し、
      仕様や設計、コードのパターンを導き出してテストを設計する
    - » 当該工程だけでなく、後工程でバグを作り込む原因となりうる部分もパターン化する
  - テスト設計だけでなく、レビューの指摘項目の設計や
    開発ガイドラインの改善にも活かすことができる
    - » テストと開発のコミュニケーションを密にするツールにもなる
    - » 自分たちの開発の弱点を得られるので、
      効果的かつ効率的にプロセス改善を行うことができる
  - テストケースあたりの不具合の検出率は向上するが、
    網羅するわけではないので品質保証には適さない
    - » 「またやっちゃった」という不具合を防ぐことができる
    - » 限られた時間で多数の不具合を検出するための手法に過ぎず、
      テスト漏れを防ぐ手法ではない
  - 不具合分析(なぜなぜ分析)のレベルが低いと、
    効果的な不具合モードを得ることができない

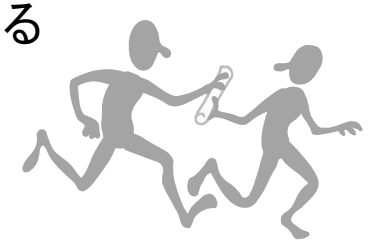


# 不具合分析のアンチパターンと効果の薄い対策

- Vaporization（その場限りの簡単なミスとして片付けてしまう）
  - コーディングミス：スキルを向上させるよう教育を行う
  - 考慮不足：しっかり考慮したかどうかレビュー時間を増やし確認する
  - うっかりミス：うっかりしていないかどうかレビュー時間を増やし確認する
- Exhaustion（不完全な実施や単なる不足を原因としてしまう）
  - しっかりやらない：しっかりやったかどうかレビュー時間を増やし確認する
  - スキル不足：スキルを向上させるよう教育を行う
  - 経験不足：経験を補うためにスキルを向上させるよう教育を行う
- Escalation（上位層に責任を転嫁してしまう）
  - マネジメントの責任：トップを含めた品質文化を醸成する
- Imposition（外部組織に責任を転嫁してしまう）
  - パートナー企業の責任：パートナー企業を含めた品質文化を醸成する
- Culturalization（文化的問題に帰着してしまう）
  - 品質意識／品質文化の欠如：全社的な品質文化を醸成する

# ⑤ 派生開発軸

- 第一世代:テストケースの再利用
  - テストケースやテスト手順書をそのまま再利用する考え方
    - » 再利用できない、再利用しても意味のないテストケースが大量に発生する
    - » 意図の分からないテストケースが多発し派生の度に肥大化していく
- 第二世代:テスト設計の再利用
  - テストケースを抽象化して再利用する考え方
    - » 派生間の仕様変更に追従しやすいので、再利用が促進される
    - » 派生開発ごとに行き当たりばったりで再利用するので、派生が続くと再利用が難しくなる
- 第三世代:再利用しやすいテスト設計
  - 再利用しやすいような抽象度を検討したうえで再利用する考え方
    - » 派生間の仕様変更に追従しやすいので、再利用が促進される
    - » 何をどのくらい再利用するかをあらかじめ見切っておく技術力が必要となる

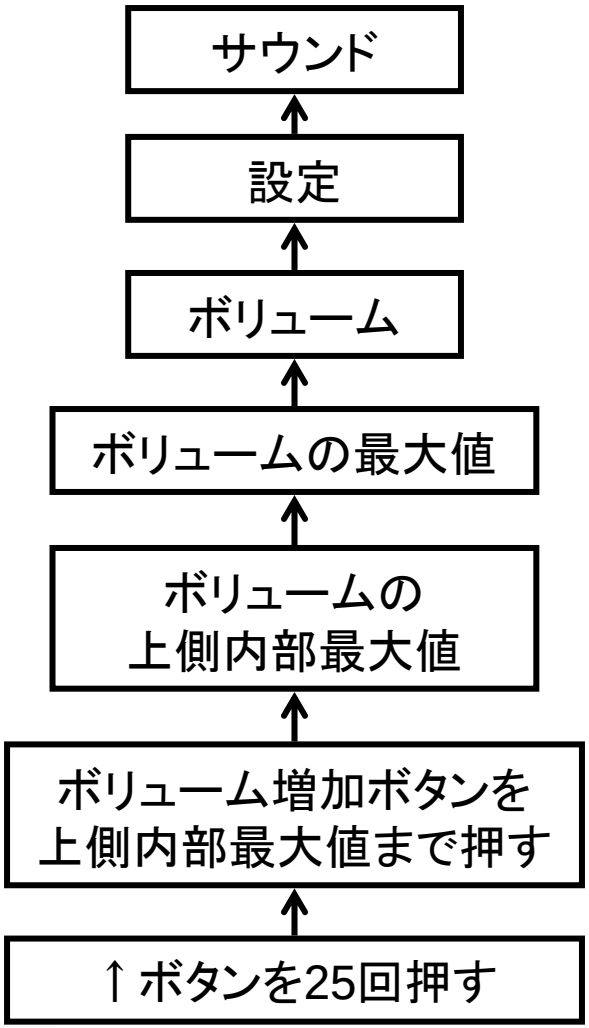


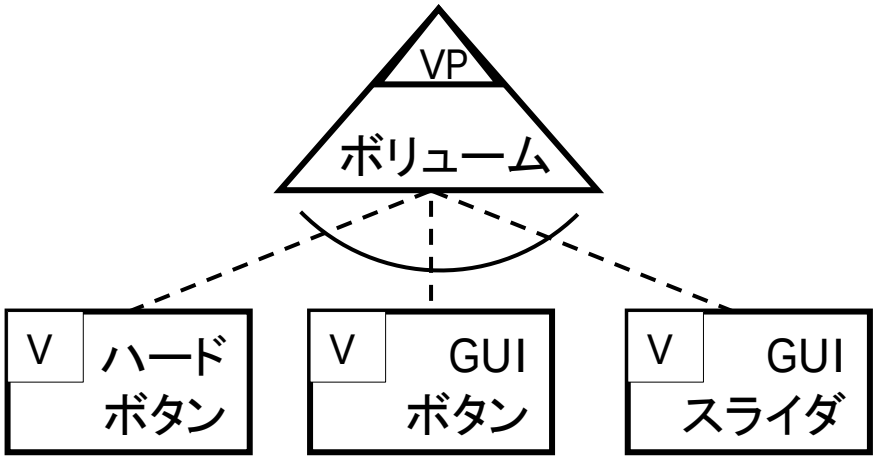
# モデリングしないとテストの再利用が難しい

| 大項目 | 中項目 | 小項目 | テスト<br>ケース |
|---|---|---|---|
| サウンド | 設定 | ボリューム | ↑ボタンを<br>25回押す |

- テストケースを再利用できるだろうか？
  - 次の製品では以下の仕様変更が起こる可能性がある
    - » ボリューム調整のUIの変更
    - » ボリューム調整の手段の増加
    - » ボリューム範囲の変更
    - » デフォルトのボリューム値の変更
  - 「25回」の意味が記述されていないので、意味不明のままテストケースを再利用する羽目になる
    - » 仕様変更に追従できず、テストケースを再設計せざるを得ない
    - » 仕様変更に追従できず、新たに増えた仕様のテストが漏れてしまう
    - » テストケースが本来の狙いを果たさなくなるため、バグを検出できない
    - » 意味不明のテストケースが増えていき、保守（削除）できなくなってしまう

# モデリングによってテストの再利用を容易にする

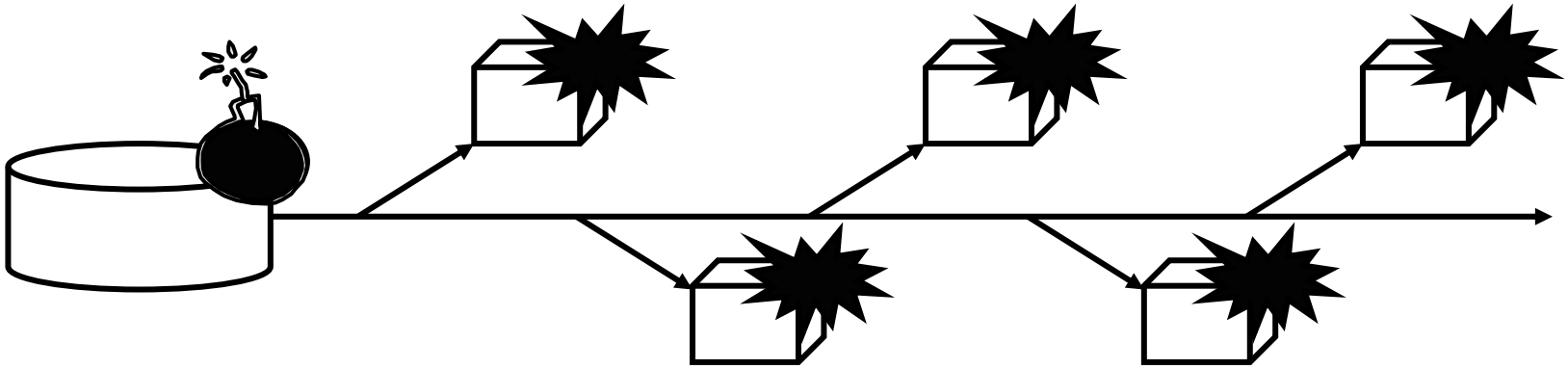



- テストモデルに「テスト設計意図」や「テスト設計理由」を入れ込むことでテストケースを再利用しやすくなる
- 上流のモデルにテストのモデルを対応させておくと、
  そもそも再利用しやすいテストを設計しておくことができる
  - 上流からテストをモデリングすると、テスト容易性を考慮するようになり分析や設計がスッキリしていく

# 派生開発・SPLEにおける頻発バグを防ぐ

- 派生開発・SPLEでは特定の種類のバグが
  派生品をまたいで頻発することがある
  - 母体に不具合モードもしくは不具合モードの芽があるため、
    派生品を開発する度にバグを作り込んでしまう
    - » 不具合モード「例外」:法規対応、多仕向地、機能複合型製品
    - » 母体の設計にそもそも難がある
    - » 仕様書群の構造そのものに不具合モードがあったりもする
  - 母体の不具合モードに着目してテストを設計することで
    頻発バグを効率的に防ぐことができる



# ⑥ テストスコープ軸 ・ ⑦ 品質軸

- 第一世代：開発対象・機能
  - 与えられた仕様書が世界の全てであるという考え方
    - » 他のサーバと接続できなかったり使いにくいシステムができる
    - » テスト技術者はテスト以外のことを考えなくてよい
- 第二世代：環境や人間を含めたシステム・品質特性
  - システムの要素同士や人間との関わりを視野に入れる考え方
    - » バグは無いし性能も高いが使われないシステムができる
    - » テスト技術者はシステムアーキテクチャや業務フローを考えなくてはいけない
- 第三世代：ビジネスやライフスタイル・価値や魅力
  - ビジネスの成功や幸せなライフスタイルという目的指向の考え方
    - » 隅々まで活用され、利益や幸せをもたらすシステムができる
    - » テスト技術者はビジネスモデルや
      ユーザの感情を考えなくてはいけない

# 例）HAYST法はどの世代か？

- 第一世代:
  - 因子と水準でテストケースを書くフォーマット
- 第二世代:
  - 禁則を回避して直交表を使い
    2因子間の組み合わせを網羅する技術
- 第三世代:
  - どういう理由で何をどう組み合わせるべきかを見抜く方法論


| | 条件A | 条件B | 条件C | 条件D | 条件E | 条件F | 条件G |
|---|---|---|---|---|---|---|---|
| テストケース1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| テストケース2 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| テストケース3 | 1 | 2 | 2 | 1 | 1 | 2 | 2 |
| テストケース4 | 1 | 2 | 2 | 2 | 2 | 1 | 1 |
| テストケース5 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
| テストケース6 | 2 | 1 | 2 | 2 | 1 | 2 | 1 |
| テストケース7 | 2 | 2 | 1 | 1 | 2 | 2 | 1 |
| テストケース8 | 2 | 2 | 1 | 2 | 1 | 1 | 2 |

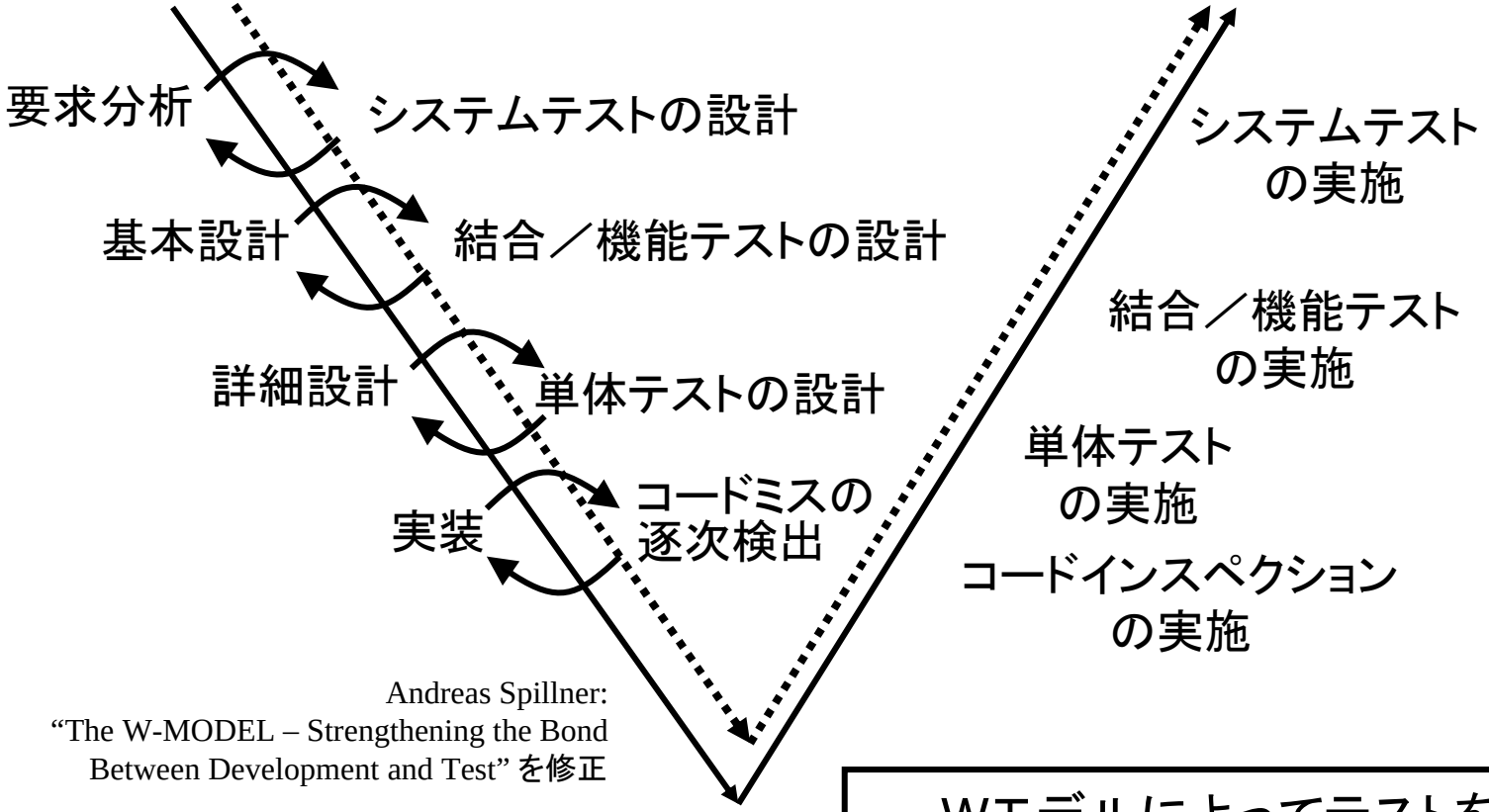
自社で行っているテストが
どの世代に属するかを考えるよりも、
どの世代の使い方で活用するか、
を考える方が重要である

# 講演の流れ

- ソフトウェアテストの世代と軸
  - ソフトウェアテストの世代
    - » 第一世代：決まっている範囲から選ぶ技術や考え方
    - » 第二世代：範囲を広げたり自ら決める技術や考え方
    - » 第三世代：全体像や本質を捉える技術や考え方
  - ソフトウェアテストの軸
    - » ① 開発ライフサイクル軸、② テストライフサイクル軸、③ テスト技術軸、④ テスト対象軸、⑤ 派生開発軸、⑥ テストスコープ軸、⑦ 品質軸
  - HAYST法はどの世代か？
- Wモデル：第三世代技術としてテストを活用するフレームワーク
  - 自工程完結と他工程配慮
  - Wモデルを支えるテスト技術
  - Wモデルのレベル
  - Wモデルの目指すべき姿

Wモデルに関する議論は、
2010年6月に開催された
ソフトウェアシンポジウム
2010のWG2での議論から
大きな薫陶を得ています。
WG2のメンバに感謝します

# Wモデル：第三世代技術として活用するフレームワーク

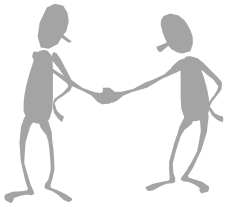



Andreas Spillner:
“The W-MODEL – Strengthening the Bond Between Development and Test” を修正

Wモデルによってテストを前倒しし、
要求や設計をスッキリさせて
そもそもバグを作り込まないようにする

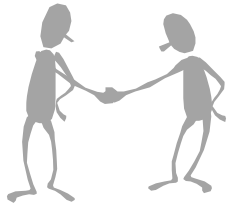
# Wモデルの基礎となる考え方：自工程完結

- 自工程完結＝自工程改善＋他工程配慮
  - 「品質を各工程で造り込み、後工程に最高品質の仕事を手渡す」ことである
    - » http://www.toyota.co.jp/jp/ir/library/annual/pdf/2008/p08_15.pdf
    - » エンジニアのポテンシャルを極限まで引き出すことに他ならない
  - 自工程完結には、自工程内だけでできる改善がある
    - » トラブルの原因の特定や改善に至るストーリーの構築は意外に難しい
  - 自工程完結には、他工程から配慮されて可能になる改善がある
    - » 分割統治や管理過多、外注過多によって視野狭窄が発生している
    - » 「きちんとやらない方がよい圧力」が働くため改善しにくい場合がある
    - » 他工程でないと測定できない／測定しにくい指標によって改善しにくい場合がある
    - » 他工程から悪さの情報をもらわないと改善しにくい場合がある
    - » どういうトラブルが発生し、どの工程でどの工程のために<br>どういう考慮をしておかなくてはいけないかが分からないと、未然防止できない
  - 他工程配慮をどのマネジメント階層で行うかは、<br>その組織のマネジメントポリシーによって異なる
    - » 欧米型組織：マネジメント層で他工程配慮を行う
    - » 日本的組織：現場で他工程配慮を行う



# Wモデルの基礎となる考え方：他工程配慮

- 他工程配慮とは
  - 品質を各工程で造り込むため、他の工程と組み合わせたり、他の工程が積極的に指標や情報を渡して改善に協力すること
    - » 上工程配慮：自工程の悪さにつながる上工程の指標や悪さを伝える
    - » 下工程配慮：下工程の悪さにつながる自工程の指標や悪さを伝える
    - » 横工程配慮：類似工程の悪さにつながる自工程の指標や悪さを伝える
  - 自分達のお客様（発注側）もパートナー（再委託先）も他工程である
  - 他工程配慮によって、開発者・開発組織の視野狭窄を乗り越える
  - 他工程配慮が成熟している組織はイノベーションを起こす可能性も高い
- 他工程配慮による融合
  - 指標的融合
    - » トレードオフと思いがちだが同時に達成できる複数指標（例：QCD）を考える
  - 技術的融合
    - » ある技術（例：テスト）を高めることで別の技術（例：設計）も高まるようなことを考える
  - 工程的・組織的融合
    - » 他工程配慮：他の組織や人のためになることを積極的に助ける仕組みを考える
  - 時間的融合
    - » 効果・効率が上がるような順序変更を考える
    - » ある時点の最適化が中長期的な最適化になるよう考える

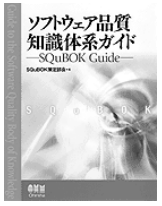


# 他工程配慮の無い現場の特徴

- 悪しき受託根性
  - 何かに従っていれば仕事をしたことになると思っているし、
    上司もそういう指示をするし、従っていれば評価が下がらない
    - » お客様や上司の言うことに従っていれば仕事をしたことになると思っている
    - » 社内の基準、プロセスに従っていれば仕事をしたことになると思っている
    - » パートナーや部下を改善する場合には基準を厳しくすればよいと思っている
- 品質とコスト・納期のトレードオフ
  - 品質は単なる開発の結果であり、
    コストや納期とトレードオフだと思っている
    - » 品質を上げようとするとコストがかかり納期が延びる、と誤解してしまっている
    - » 上流はバグを作ってしまうのが当然だと思っており、
      レビューやテスト、監査でバグを見つけることでしか品質を上げられないでいる
    - » 本質安全よりもまず機能安全を考えてしまう
- 安直な分割統治
  - 異なる技術を専門家毎に高度化しようと思っている
    - » 設計とテスト、PMとSEPG、開発と教育
    - » Vモデル、コストダウンのための第3者検証

# 他工程配慮を支える考え方：TQM/SQuBOK

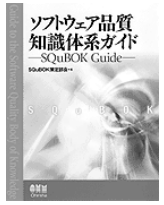
- 「品質」
  - 仕事の質，サービスの質，
    情報の質，工程の質，部門の質，
    人の質，システムの質，
    会社の質など，
    これら全てを含めた「質」
- 「品質保証」
  - お客様に安心して使って頂けるような製品を提供するためのすべての活動
    - » プロダクトとプロセスが、特定された要求に合致しているかどうかという十分な確信を提供する活動ではない




- 「改善」
  - 不十分でもとにかく動き出して
    全員が今より高いところを
    目指してプロセスそのものを
    改善しながら進める
    - » 欧米流の、プロセスを定義して
      その通りに実行していることを
      確認することではない
- 「品質第一」
  - 品質を高めることによって
    手戻りや作業のムダを省き，
    継続的な納期の短縮や
    コストダウンにつなげていく
    - » 納期を厳守するために
      必要な作業を省くのではない

# 他工程配慮を支える考え方：TQM/SQuBOK

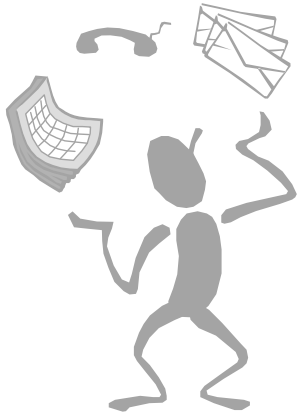
- 「品質の作り込み」
  - より上流で“悪さ”の知識を子細に活用し障害を排除していく
    - » ただ単にきちんと作る、という意味ではない
- 「次工程はお客様」
  - 他の工程を助けるような改善を進め，全体最適へと向かっていく
- 「人間性尊重」
  「組織活性化」
- 「小集団活動」
- 「5ゲン主義」
  - 現場・現物・現実
  - 原理・原則




- 「全員参加」
  - 品質管理はスタッフだけではなく、現場も経営陣も一丸となって進めなければならない
  - 現場の関係者全員が納得してベクトルをあわせ，目標達成のために取り組む
  - 現場中心のため隅々まで改善が行き届くとともに，全員が自ら考えて行動する組織を構築できる

# 最後にテストで抑えようとしてもムリである

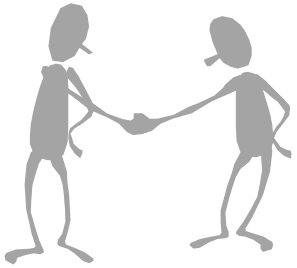
- 成熟度の低い組織は、下流のテストで何とかしようとする
  - テストにかかる人員や工数を増やそうとする
  - 上流の品質が低いと、テストを増やすと混乱も増幅される
  - 上流を考えずにテストの品質だけを上げようとすると、テストは無限になる
  - テストを増やしすぎて、上流を減らしすぎてしまいがちになる
    - » さらに上流の品質が下がり、テストが混乱する
  - 結果として、増えた人員が品質に寄与せず、マッチポンプに問題を解決するだけになってしまう



- 上流から品質を作り込むのは常識である
  - だから構造化やOO、MBDを導入しているのだ
  - 形式手法を導入したいのだ
  - しかし、どんな「銀の弾丸」を導入しても、結局のところパーキンソンの法則が働いてしまう
  - 「上流からの品質の作り込み」の本質は、ノウハウの循環の活性化である

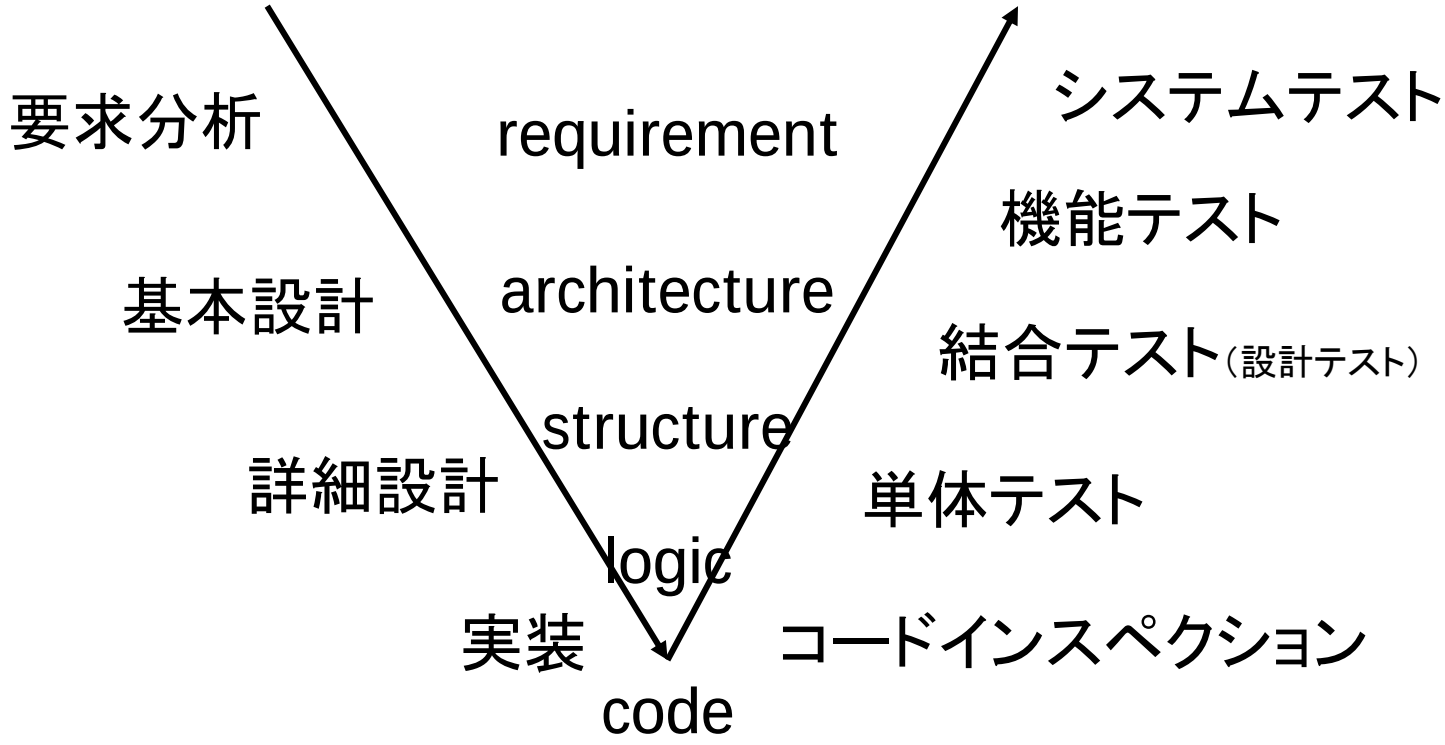
# 上流と協調し、上流を改善するようにテストする

- 成熟度の高い組織は、テストと上流が協調している
  - 取りあえずテストの人員や工数を増やそうとするのではなく、自分たちのテストのムダやモレをしっかり分析しようとする
  - その結果、わざわざ下流まで待たなくても検出できる不具合がかなりあることが分かる
  - ビルドして初めてテストを考えるのではなく、上流からテスト設計を行っていくことでバグを防ぐ
    - » 上流からのテストは、単なる要求の裏返しではない
  - 上流からテストの品質を上げようとすると、テストは無限にならず、出荷時の品質リスクも考えられるようになる
  - 「上流からの品質の作り込み」の本質は、ノウハウの循環の活性化である



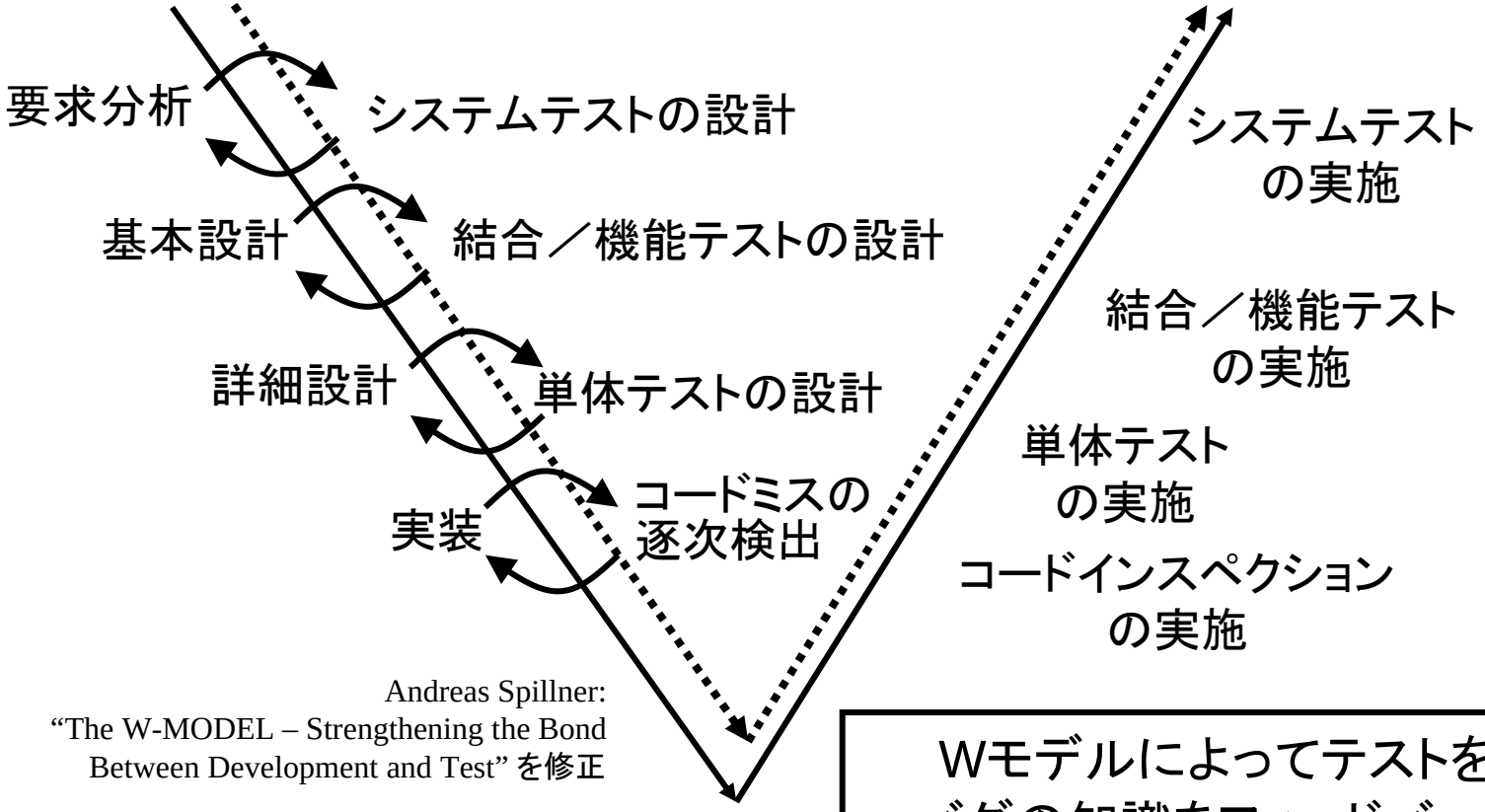
- 「見通しのよい」開発ができるようになる
  - 品質の高いソフトを開発している組織は、自分たちがやっていることの意味合いや位置づけ、全体像を把握し、どうなるかを概ね推測できる
  - 開発やマネジメント/プロセスだけでなく、レビューやテストなど評価系の意味合いや位置づけ、全体像をしっかり把握する必要がある

# 分割統治的プロセスの例：Vモデル




Vモデルでは上流とテストが分断され、
バグの知識がフィードバックされない

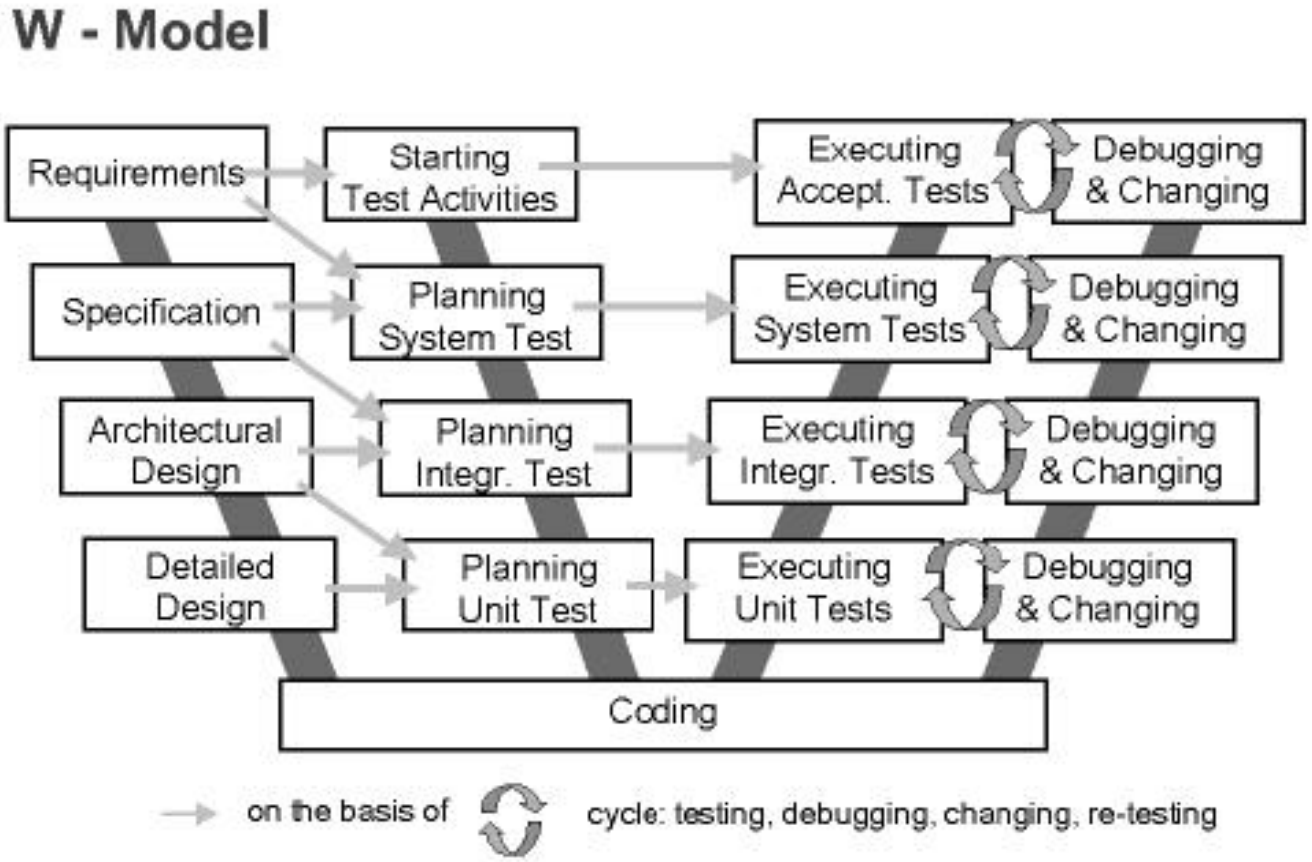
# 他工程配慮的プロセスの例：Wモデル




Andreas Spillner:
“The W-MODEL – Strengthening the Bond Between Development and Test” を修正

Wモデルによってテストを前倒しし、
バグの知識をフィードバックすることで
要求や設計をスッキリさせて
そもそもバグを作り込まないようにする

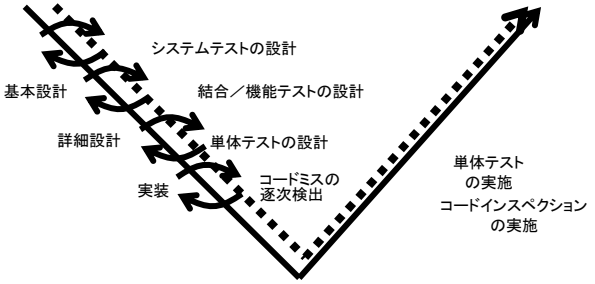
# Wモデル: テストと上流が協調した開発プロセス




Andreas Spillner:
"The W-MODEL – Strengthening the Bond Between Development and Test"
STAREAST2002

# Wモデルを支えるテスト技術

- 基本的なテスト技術
  - 体系的なテスト技法の活用、期待結果の導出
  - 網羅テストの段階的詳細化
  - テストレベルの設計もしくは解釈
- 進んだテスト技術
  - 組み合わせテスト設計、Critical Combination Analysis、禁則分析
  - テスト観点の抽出・整理・体系化
  - 不具合モード分析・後工程不具合リスク分析
  - リスクベースドテスト
  - グレーボックステスト
  - 探索型テスト
- 開発全般に関わるテスト技術
  - テスト容易性設計
  - 出荷判定基準の明確化




テストの改善をきっかけにして、
そもそもバグを作り込みにくい
開発プロセスに改善していく

# Wモデルの効果:タイプ1

- 単純に作業順序を入れ替えるだけだが、
  上流でテストを設計することで細かく考えることができ、
  抜け漏れ矛盾を減らすことができる
  - テスト設計をきちんと行うことで検出できる不具合を、
    下流まで遅らせることなく上流で検出することにより、
    不具合の伝播や増殖、不要な設計変更などのムダを省くことができる
    - » 開発者は開発時に主要な仕様や設計を考えるに留まり、
      実は細かくは考えていないことが多い
    - » 仕様や設計が複数の組織やバージョンに由来し、整合性を取れていないことがある
    - » テストに頼る風土の組織では、一貫性すらきちんと確認していないことがある
- テスト設計の質を少しだけ向上することができる
  - 仕様や設計を行った直後にテストを設計するような手順にすると、
    極めて単純なテストの抜け漏れだけは防ぐことができる
    - » テストの抜け漏れが何でも防げるわけではない

# Wモデルの効果:タイプ1の注意点

- テスト設計をきちんと行っていない組織では効果が出ない
  - アドホック型やコピー＆ペースト＆モディファイ型のテスト設計では、
    テスト設計時に不具合が検出できることが少ないため、
    テスト設計を上流にシフトしても効果は少ない
    - » 何をどれくらい網羅し、どういう理由でどのくらい間引くかを
      明示していない組織では、抜け漏れ矛盾を見つけることができない
- レベル1に留まったままだと逆に問題が起きやすい
  - 開発上流で膨大なテストケースを記述することになるため、
    仕様変更・設計変更時に膨大なテストケースの書き直し作業が必要となり、
    工数超過を起こしてしまう
    - » 仕様の書き方をそもそも改善しないといけない
    - » テストケース生成ツールで自動生成させたくなるが、
      自動生成させると抜け漏れ矛盾が見つかりにくくなる
    - » 結局のところ「テスト設計とは何か」をきちんと考えないといけない

# Wモデルの効果:タイプ2

- 開発組織がいま持っている観点できれいに作る（汚く作らせまいとする）よう圧力をかけることができる
  - 開発者は「もっとすっきりさせたいが時間がない」というジレンマを抱えている
    - » 上流で（組み合わせ）テスト項目数を概算することで「もっとすっきりさせないと、テストで膨大な時間がかかる」と圧力をかけることができる
  - （組み合わせ）テスト設計を上流で行うことで、仕様や設計、コードがすっきりし、不具合を減らすことができる
    - » 組み合わせテスト設計で結合度の低下や依存関係の存在をきちんと把握するので開発時に予期していない副作用（組み合わせ不具合の芽）を防ぐことができる
    - » 不必要な依存関係や禁則を持った仕様を減らすことができる
  - テスト容易性を検討でき、作り込むことができる
    - » テスト設計しやすい仕様・設計・コードは、可読性が高く複雑度が低いうえに、ふるまいを予測しやすい
- テスト設計の質を向上することができる
  - すっきり作ってあれば依存関係を追いやすく例外事項が少ないため、工数を増やさずに抜け漏れを少なくできる

# Wモデルの効果:タイプ3

- 開発組織がいま持っていないような
  多面的な観点から気をつけて作ることができる
  - 開発者はあらゆる観点から検討しているようで、
    採用した開発のやり方に固有の観点抜けが出てしまう
    - » 例)制御系の開発では状態遷移の検討の抜けが発生しやすい
  - テスト設計の際に観点を俯瞰的・多面的に列挙し整理することで、
    開発者が気づいていない観点を示唆することができる
    - » テスト技術者は過去の不具合の分析などにより、
      抜けやすい観点を知っている場合がある
    - » 開発者は開発対象に必要な観点を絞り込むように頭を使うが、
      テスト技術者はテスト対象を俯瞰的・多面的に捉えようとする
  - 必ずしもテストケースを詳細に記述する必要はない
- テスト設計の質を向上することができる
  - 開発時に検討した観点をテストで共有することで、
    特に内部構造に関する

# Wモデルの効果:タイプ4

- 開発組織が忘れがちな、要求分析・設計・実装工程の「由来」と「行く末」を常に意識し、両者の乖離を防ぐことができる
  - 開発者は意外に「なぜそういう設計をしているのか」や「こういう設計変更を行うとシステム全体としてはどういう振る舞いをするのか」などを考えずに開発しているため、「由来」と「行く末」が乖離してしまう
  - テスト設計の際に網羅型テストの段階的詳細化を行うことにより、由来(要求や制約、根拠)をイメージバックしながら開発できるようになる
    - » どのような要求群から成り立つのか、どのような制約がありうるのか、などを常に考えながら作るようになる
  - テスト設計の際に出荷判定基準の明確化や期待結果の導出を行うことにより、行く末(出荷基準やトリアージ順位、ふるまい)をイメージフォワードしながら開発できるようになる
    - » どのような出荷基準になっているのか、どのようなトリアージ順位になっているのか、などを常に考えながら作るようになる
  - 由来と行く末が乖離する開発に歯止めをかけることができるようになる
- テスト設計の質を向上することができる
  - テスト目的を常に意識するため、心配だからという理由のテストが減っていく

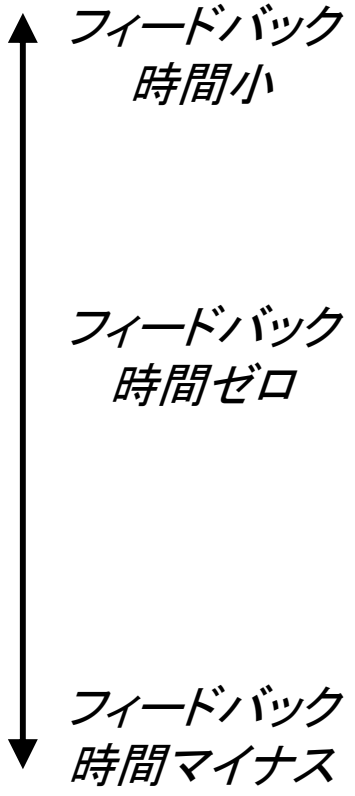
# Wモデルの効果:タイプ5

- 開発組織の持っている弱点を明確にし防ぐことができる
  - 下流で不具合モード分析を行って当該工程の弱点をフィードバックしてもらうことにより、頻発する不具合を防止した開発ができるようになる
    - » 下流での不具合モード分析により、
      自分たちが作り込みやすい不具合のパターンや
      ドメイン特有の作り込みやすい不具合のパターンを把握し、
      最初から気をつけて開発することができる
    - » 「上流からの品質の作り込み」とは、
      必要なノウハウを必要な時期に必要なだけ駆使することであり、
      最初から矛盾の無いものを作って矛盾の無いように自動生成することではない
  - 上流で後工程不具合リスク分析を行って、上流で気付いていた「心配事」を当該工程にフィードフォワードしてもらうことにより、
    分かっていたのに起きてしまった検討漏れを減らすことができるようになる
    - » 上流での後工程不具合リスク分析により、
      工程が進んだ後に不具合になる
      リスキーな仕様・設計・コードの情報をもらうことができ、
      リスキーな部分に常に気をつけて開発することができる
- テスト設計の質を向上することができる
  - 不具合が作り込まれている確率の高いところからテストすることができる

# Wモデルの進化：フィードバック時間の減少

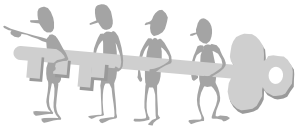
- 各タイプの効果をあげるように
  テスト技術を進化させ前倒しすることで
  開発上流から品質の作り込みを行っていく

1. 開発者が開発（仕様策定、設計、実装）を行った後で、
   すぐにテスト技術者がテスト設計を行いフィードバックする
2. 開発者が開発を行った後で、
   すぐに開発者が自らテスト設計を行いフィードバックする
3. 開発者が開発を行うと同時に、
   テスト技術者が独立にテスト設計を行いフィードバックする
4. 開発者が開発を行うと同時に、
   テスト技術者が開発者とワイガヤしフィードバックしながら
   テスト設計を行う
5. 開発者が開発を行いながら、
   開発者が自らテスト設計を行い両者を同時に改善していく
6. 開発者が開発を行う前にテストについても思索を深め、
   はじめからバグの無い開発を行う



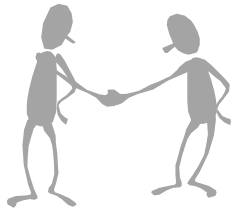
# Wモデルの目指すべき姿

- 全ての工程でノウハウを生みだし共有し駆使する開発プロセス
  - ノウハウの抽出・蓄積・共有・駆使・改訂がプロセスに埋め込まれるようになる
  - 開発チームが開発進行中に自分達でプロセス改善をするようになる
  - 幅と遠さの両方で、開発者の見通しをよくすることができるようになる
    - » 開発者は一般的に分割統治・詳細隠蔽の原則で発想することが多いので、テスト設計者と一緒に設計検討をすると見通しがよくなることが期待できる
  - ソフトウェア開発者の多能工化が進む
- 開発者が「最もよいやり方で考える」ことに近づけていくプロセス
  - テストについて深く洞察することで、テストを実施することなく品質を高められるようになる
  - 開発者の気づきのフィードバックサイクルが極限まで小さくなり、開発予測力が高まっていく
    - » リスクベースドテストをきちんと運用できる技術力が備わっていく
  - よい開発をするという点において、開発者とテスト技術者のゴールは同じである



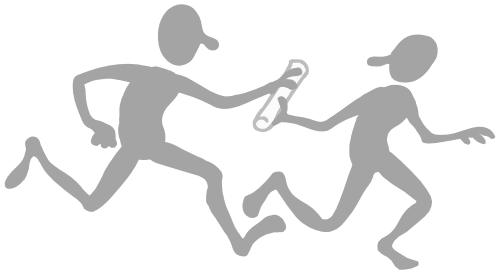
# Wモデルの導入へのアプローチ

- プロセス“改革”アプローチ
  - スタッフ部門や上級管理職が主導でWモデルのプロセスマニュアルを作り、各部門に展開して導入する
    - » 私見だが、いきなり“改革”を現場に押しつけても多分うまくいかない
    - » プロセスの“改善”をextremeに行った結果、プロセスイノベーションが起きただけである
- レビュー強化アプローチ
  - レビューを強化しようという運動として導入する
    - » 私見だが、レビューの指摘項目の設計の強化にはつながらず、結局プロセスをいじって終わってしまうと思う
- 工程順序変更アプローチ
  - テスト設計を改善し、テスト設計時に不具合を見つけてもらうことで、下流のテスト設計まで不具合検出を遅らせるムダを意識してもらい、工程順序を変更する圧力を自発的に発生させる
    - » 定着しやすいが時間がかかる
- 設計ノウハウ蓄積・活用アプローチ
  - 設計者同士のノウハウ共有という運動として不具合モードを蓄積し、それを全工程で活かすようにする
    - » すぐに実行できるが、開発者が忙しいと停滞しやすい



# Wモデルの導入へのアプローチ

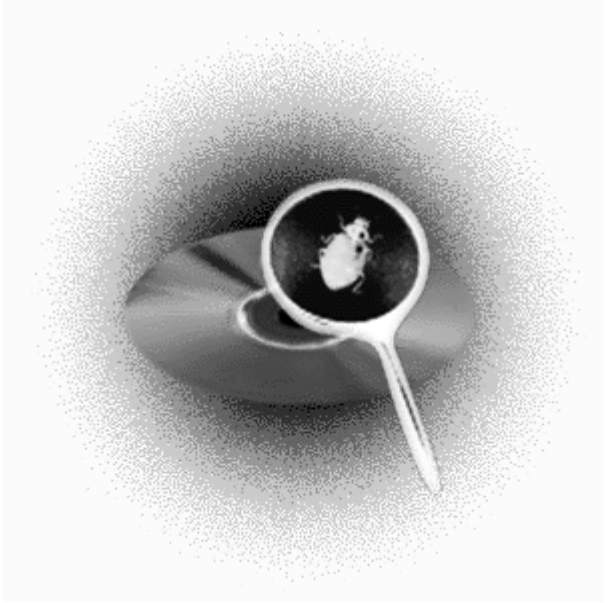
- 技術の発展は癒着→剥離→融合の段階を経ることを意識し、剥離させずに融合させようとしても定着しないことを理解する
  - 自分達は何を分かっており何を分かっていないのかをきちんと理解したうえで、適切な技術やノウハウを駆使できるようにする
    - » 癒着：個々の技術を意識せず、したがって技術成熟度が上がらないまま、様々な技術を暗黙的かつKKDで組み合わせて利用している状態
    - » 剥離：個々の技術を意識して区別することで技術成熟度を上げる状態
    - » 融合：複数の技術を意識的に組み合わせることでムダを省き全体最適を実現している状態
- 自分たちがきちんと仕事をすれば責任は無い、というカルチャーやコンセプトで導入すると、Wモデルは順序入れ替え以上の効果を及ぼさない
  - Wモデルは（静的な）プロセスモデルというよりも、プロセス進化のモデルであることを理解する
    - » 次工程はお客様、TOC、自工程完結などのコンセプトの理解が重要となる



# まとめ

- ソフトウェアテストは第三世代に突入している
  - 第一世代：決まっている範囲から選ぶ技術や考え方
    - » 第一世代技術であっても、技術力を高めて
      きっちり使いこなせば品質は確保できる
  - 第二世代：範囲を広げたり自ら決める技術や考え方
    - » 「説明無責任」にならないように、
      自分達の開発力をきちんと把握しなくてはならない
  - 第三世代：全体像や本質を捉える技術や考え方
    - » テストの質を高めることで上流の質が高まり、
      上流の質が高まることでテストの質も高まる
- 自組織に合った世代でテストを効果的に行い、
  さらに先の世代の使い方にチャレンジし進化して欲しい
  - そのためには、テスト技術だけでなく
    開発技術もプロセスもマネジメントも文化もすべて高度化するよう
    チャレンジを続けなくてはならない

# ご清聴ありがとうございました



電気通信大学　西 康晴
http://blues.se.uec.ac.jp/
nishi@se.uec.ac.jp